Safepoints in HotSpot JVM

Term Stop-the-World pause is usually associated with garbage collection. Indeed GC is a major contributor to STW pauses, but not the only one.

Safepoints

In HotSpot JVM Stop-the-World pause mechanism is called safepoint. During safepoint all threads running java code are suspended. Threads running native code may continue to run as long as they do not interact with JVM (attempt to access Java objects via JNI, call Java method or return from native to java, will suspend thread until end of safepoint).

Stopping all threads are required to ensure what safepoint initiator have exclusive access to JVM data structures and can do crazy things like moving objects in heap or replacing code of method which is currently running (On-Stack-Replacement).

How safepoints work?

Safepoint protocol in HotSpot JVM is collaborative. Each application thread checks safepoint status and park itself in safe state in safepoint is required.

For compiled code, JIT inserts safepoint checks in code at certain points (usually, after return from calls or at back jump of loop). For interpreted code, JVM have two byte code dispatch tables and if safepoint is required, JVM switches tables to enable safepoint check.

Safepoint status check itself is implemented in very cunning way. Normal memory variable check would require expensive memory barriers. Though, safepoint check is implemented as memory reads a barrier. Then safepoint is required, JVM unmaps page with that address provoking page fault on application thread (which is handled by JVM's handler). This way, HotSpot maintains its JITed code CPU pipeline friendly, yet ensures correct memory semantic (page unmap is forcing memory barrier to processing cores).

When safepoints are used?

Below are few reasons for HotSpot JVM to initiate a safepoint:

Garbage collection pauses
Code deoptimization
Flushing code cache
Class redefinition (e.g. hot swap or instrumentation)
Biased lock revocation
Various debug operation (e.g. deadlock check or stacktrace dump)

Trouble shooting safepoints

Normally safepoints just work. Thus, you can care less about them (most of them, except GC ones, are extremely quick). But if something can break it will break eventually, so here is useful diagnostic:

-XX:+PrintGCApplicationStoppedTime – this will actually report pause time for all safepoints (GC related or not). Unfortunately output from this option lacks timestamps, but it is still useful to narrow down

problem to safepoints.
-XX:+PrintSafepointStatistics –XX:PrintSafepointStatisticsCount=1– this two options will force JVM to report reason and timings after each safepoint (it will be reported to stdout, not GC log).

References

·How does JVM handle locks – quick info about biased locking

·HotSpot JVM thread management

16 comments:

Nice article Alexey. Do you know what part of the JDK code really does this - "JVM unmaps page with that address provoking page fault on application thread"?

I think the Azul JVM also used to do this to quickly trap moved/GC'ed addresses.

ReplyDelete

Replies

- Using page faults for read barrier ("quickly trap moved/GC'ed addresses") would be prohibitively expensive. Azul JVM does not use page faults for read barrier, though it is using this technique for defragmenting physical memory associated with large object.

  Azul is using custom page mapping to facilitate software read barrier, but this technique does not relay on page faults.

  Or at least it was that way last time I was working with Azul.

Delete

Reply

Thank you for an eye-opening article on safepoints. Do you know if there is any way to identify the reason for a huge pause of hundreds of seconds that does not appear to be related to GC activity?

```
Total time for which application threads were stopped: 0.0020916 seconds
Total time for which application threads were stopped: 0.0677614 seconds
Total time for which application threads were stopped: 0.0016208 seconds
Total time for which application threads were stopped: 195.2580105 seconds
Total time for which application threads were stopped: 0.0313111 seconds
Total time for which application threads were stopped: 0.0005465 seconds
Total time for which application threads were stopped: 0.0006269 seconds
```

ReplyDelete

Replies

- First enable safe point logging -XX:+PrintSafepointStatistics -XX:PrintSafepointStatisticsCount=1
  This will allow you to understand whenever safepoint is culprit.

  Last problem with slow safepoints, was bug in JIT combined with weird application code.

  Trying latest JVM is another step.

Delete

Reply

We switched to 1.6.0_43, at the time that happened we had 1.6.0_31. One of the reasons was bug 2221291. Can you tell me the bug ID for the problem related to JIT?

ReplyDelete

Replies

- No, I didn't track exact bug. Slight change of code has solved issue in my case.
  Yep, 2221291 is a nasty one.

  Delete

Reply

Very informative article, thank you. We have seen due to IO overload inside Linux. When this happens, GC log entries show use_time ~0, sys_time ~ 0, and real-time ~ at least 1 second. We are able to recreate this type of stalls in the lab too. It turns out that deferred writes to append a file can be blocked for a long time when the write is blocked by journal commit. Or when dirty_ratio is exceeded. We straced the Java process and could correlate some but not all of the stalls to GC threads when they write to the gc.log file. If GC threads do not have park the Java threads running in kernel mode, we are stumped about what else could have caused the stall (where user_time ~ 0, sys_time ~0). Any other data/traces you would recommend to help us understand the issue better? Many thanks.

ReplyDelete

Replies

- Have you enabled -XX:+PrintSafepointStatistics ?

  Sometimes I've seen JVM spending too much time trying to enter to safe point. Safe point initiation time is accounted to GC pause time.

  Another suspects are native threads taking GC lock via JNI (+XX:+PrintJNIGCStals may help to identify if this is a case).

  Regards,
  Alexey

  Delete

Reply
Hi Alexey, thanks for the feedback. We did not always turn on -XX:+PrintSafepointStatistics because the output is so obscure. Our test program that recreates the stall just uses log4j and does not make calls to JNI but it's great to know about this option PrintJNIGCStalls.

ReplyDelete
AnonymousNovember 14, 2013 at 10:17 AM

Hi Alexey, your articles are very informative. Recently i have faced a situation where GC is taking mamooth time and not sure what can be the reason. Here is the output of jstat -gc command

S0C S1C S0U S1U EC EU OC OU PC PU YGC YGCT FGC FGCT GCT

77440.0 73088.0 22896.4 0.0 1946624.0 222690.4 4194304.0 3638965.1 262144.0 216641.1 1093 11258.452 3 10031.493 21289.944

ReplyDelete

Replies

- Hi,

  To be able to give you a reasonable advise, I need
  - your JVM start parameters
  - excerpt from your GC logs with at least -Xx:+PrintGCDetails enabled

  I would also suggest you to post question on stackoverflow.com (and post link here) as it is better platform for that kind of questions.

  Regards,
  Alexey

  Delete

Reply
AnonymousNovember 14, 2013 at 1:34 PM

Hi Alexey, as suggested by you i have posted question on stackoverflow.com and here is the link

http://stackoverflow.com/questions/19979030/gc-is-taking-mammoth-time

Also here is the start up parameters and PrintGCDetails are not enabled and will take time as it is production server.

-Xms6144m -Xmx6144m -XX:MaxPermSize=256m -Djava.security.policy=/bea/wlserver_10.3/server/lib/weblogic.policy -Dweblogic.ProductionModeEnabled=true -da -

Dplatform.home=/bea/wlserver_10.3 -
Dwls.home=/bea/wlserver_10.3/server -
Dweblogic.home=/bea/wlserver_10.3/server -
Dweblogic.management.discover=true -Dwlw.iterativeDev=false -
Dwlw.testConsole=false -Dwlw.logErrorsToConsole=false -
Dweblogic.ext.dirs=/bea/patch_wls1036/profiles/default/sysext_manifes
t_classpath -Djava.awt.headless=true -Djava.net.preferIPv4Stack=true -
Duser.timezone=GMT -Dfile.encoding=UTF-8 -Duser.language=en -
Duser.country=US -Dweblogic.wsee.wstx.wsat.deployed=false -
XX:+DisableExplicitGC

ReplyDelete
AnonymousNovember 14, 2013 at 3:51 PM

Running VMStat for 5 hours has given the following result, i am providing a part of the output:
swap free re mf pi po 40336468 4025208 383 5473 465 59 40336132 4025732 383 5477 465 59 40336020 4025732 383 5478 465 59 40335940 4025752 383 5479 465 59 40335860 4025776 383 5479 465 59 40335776 4025796 383 5480 465 59 40335696 4025816 383 5481 465 59 40335584 4025816 383 5482 464 59 40335504 4025836 383 5483 464 59 40335420 4025856 383 5484 464 59

Can we inference something from this output

ReplyDelete
Hi,

I am getting millions of following messages:
54.104: ThreadDump [153 2 3] [0 0 0 0 0] 0
vmop [threads: total initially_running wait_to_block] [time: spin block sync cleanup vmop] page_trap_count
54.104: ThreadDump [153 3 4] [0 0 0 0 0] 0

```
vmop [threads: total initially_running wait_to_block] [time: spin block
sync cleanup vmop] page_trap_count
54.104: ThreadDump [ 153 1 6 ] [ 0 0 0 0 0 ] 0
vmop [threads: total initially_running wait_to_block] [time: spin block
sync cleanup vmop] page_trap_count
54.104: ThreadDump [ 153 2 2 ] [ 0 0 0 0 0 ] 0
vmop [threads: total initially_running wait_to_block] [time: spin block
sync cleanup vmop] page_trap_count
54.105: ThreadDump [ 153 0 2 ] [ 0 0 0 0 0 ] 0
vmop [threads: total initially_running wait_to_block] [time: spin block
sync cleanup vmop] page_trap_count
54.105: ThreadDump [ 153 1 6 ] [ 0 0 0 0 0 ] 0
vmop [threads: total initially_running wait_to_block] [time: spin block
sync cleanup vmop] page_trap_count
```

What could be the reason of ThreadDump activity in SafePoint?

ReplyDelete

I would guess, it is result of profiler. Thread dumps are widely used by profiler and sometimes by monitoring tools. Java code could also cause thread dump for itself.

ReplyDelete

Replies

- Thanks for your quick updates.

  There are no profiler attached to the java process.

  I don't see any log of general Threaddumps it means they are Internal Thread dumps as you mentioned.

  Any other suggestions what to look for next?

Thanks,
KEshav

Delete

Reply

Links to this post

Create a Link